Voice Activity Detector (VAD) Algorithm User's Guide

SPIRIT CORP
DSP Software Source
www.spiritDSP.com/CST

Literature Number: SPRU635
March 2003

TEXAS INSTRUMENTS

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Mailing Address:

Texas Instruments
Post Office Box 655303
Dallas, Texas 75265

Preface

Read This First

About This Manual

The following abbreviations are used in this document:

AGC	Automatic Gain Control
CNG	Comfort Noise Generator
VAD	Voice Activity Detector
XDAIS	TMS320 DSP Algorithm Standard

Related Documentation From Texas Instruments

Using the TMS320 DSP Algorithm Standard in a Static DSP System (SPRA577)

TMS320 DSP Algorithm Standard Rules and Guidelines (SPRU352)

TMS320 DSP Algorithm Standard API Reference (SPRU360)

Technical Overview of eXpressDSP-Compliant Algorithms for DSP Software Producers (SPRA579)

The TMS320 DSP Algorithm Standard (SPRA581)

Achieving Zero Overhead with the TMS320 DSP Algorithm Standard IALG Interface (SPRA716)

Related Documentation

Automatic Gain Control for Voice Applications, Spirit Corp., 2002

Comfort Noise Generator for Voice Applications, Spirit Corp., 2002

Trademarks

TMS320™ is a trademark of Texas Instruments.

SPIRIT CORP™ is a tradmark of Spirit Corp.

All other trademarks are the property of their respective owners.

Software Copyright

If You Need Assistance . . .

❑ World-Wide Web Sites

TI Online	http://www.ti.com
Semiconductor Product Information Center (PIC)	http://www.ti.com/sc/docs/products/index.htm
DSP Solutions	http://www.ti.com/dsp
320 Hotline On-line™	http://www.ti.com/sc/docs/dsps/support.htm
Microcontroller Home Page	http://www.ti.com/sc/micro
Networking Home Page	http://www.ti.com/sc/docs/network/nbuhomex.htm
Military Memory Products Home Page	http://www.ti.com/sc/docs/military/product/memory/mem_1.htm

❑ North America, South America, Central America

Product Information Center (PIC)	(972) 644-5580		
TI Literature Response Center U.S.A.	(800) 477-8924		
Software Registration/Upgrades	(972) 293-5050	Fax: (972) 293-5967	
U.S.A. Factory Repair/Hardware Upgrades	(281) 274-2285		
U.S. Technical Training Organization	(972) 644-5580		
Microcontroller Hotline	(281) 274-2370	Fax: (281) 274-4203	Email: micro@ti.com
Microcontroller Modem BBS	(281) 274-3700 8-N-1		
DSP Hotline		Email: dsph@ti.com	
DSP Internet BBS via anonymous ftp to ftp://ftp.ti.com/pub/tms320bbs			
Networking Hotline		Fax: (281) 274-4027 Email: TLANHOT@micro.ti.com	

❑ Europe, Middle East, Africa

European Product Information Center (EPIC) Hotlines:		
Multi-Language Support	+33 1 30 70 11 69	Fax: +33 1 30 70 10 32
Email: epic@ti.com		
Deutsch	+49 8161 80 33 11 or +33 1 30 70 11 68	
English	+33 1 30 70 11 65	
Francais	+33 1 30 70 11 64	
Italiano	+33 1 30 70 11 67	
EPIC Modem BBS	+33 1 30 70 11 99	
European Factory Repair	+33 4 93 22 25 40	
Europe Customer Training Helpline		Fax: +49 81 61 80 40 10

❑ Asia-Pacific

Literature Response Center	+852 2 956 7288	Fax: +852 2 956 2200
Hong Kong DSP Hotline	+852 2 956 7268	Fax: +852 2 956 1002
Korea DSP Hotline	+82 2 551 2804	Fax: +82 2 551 2828
Korea DSP Modem BBS	+82 2 551 2914	
Singapore DSP Hotline		Fax: +65 390 7179
Taiwan DSP Hotline	+886 2 377 1450	Fax: +886 2 377 2718
Taiwan DSP Modem BBS	+886 2 376 2592	
Taiwan DSP Internet BBS via anonymous ftp to ftp://dsp.ee.tit.edu.tw/pub/TI/		

❑ Japan

Product Information Center	+0120-81-0026 (in Japan)	Fax: +0120-81-0036 (in Japan)
	+03-3457-0972 or (INTL) 813-3457-0972	Fax: +03-3457-1259 or (INTL) 813-3457-1259
DSP Hotline	+03-3769-8735 or (INTL) 813-3769-8735	Fax: +03-3457-7071 or (INTL) 813-3457-7071
DSP BBS via Nifty-Serve	Type “Go TIASP”	

❑ **Documentation**

When making suggestions or reporting errors in documentation, please include the following information that is on the title page: the full title of the book, the publication date, and the literature number.

Mail: Texas Instruments Incorporated
Technical Documentation Services, MS 702
P.O. Box 1443
Houston, Texas 77251-1443

Email: dsph@ti.com Email: micro@ti.com

Note: When calling a Literature Response Center to order documentation, please specify the literature number of the book.

For product price & availability questions, please contact your local Product Information Center, or see www.ti.com/sc/support http://www.ti.com/sc/support for details.

For additional CST technical support, see the TI CST Home Page (www.ti.com/telephonyclientside) or the TI Semiconductor KnowledgeBase Home Page (www.ti.com/sc/knowledgebase).

If you have any problems with the Client Side Telephony software, please, read first the list of Frequently Asked Questions at http://www.spiritDSP.com/CST.

You can also visit this web site to obtain the latest updates of CST software & documentation.

Contents

Figures

Tables

Notes, Cautions, and Warnings

Chapter 1

Introduction to Voice Activity Detector (VAD) Algorithms

This chapter briefly describes the Voice Activity Detector algorithms and related products used with the TMS320C5400 platform.

For the benefit of users who are not familiar with the TMS320 DSP Algorithm Standard (XDAIS), brief descriptions of typical XDAIS terms are provided.

Topic **Page**

1.1 Introduction

This document describes the implementation of the Voice Activity Detector (VAD) developed by SPIRIT Corp. for TMS320C5400 platform and intended for integration into various embedded devices such as:

- vocoders
- answering machines
- speech recorders
- VoIP systems
- PBX equipment

SPIRIT VAD detects the presence of speech in the signal. It has special adaptive algorithm to automatically adjust to the level of the noise in the signal, in order to provide robust operation even in the noisy speech. It has many user configurable parameters, allowing the algorithm to optimally tune itself for a specific application. VAD also outputs several coefficients (up to 10) that characterize spectral envelope of the noise (when no speech is detected), so that the regenerated noise would be similar to the original noise.

The SPIRIT VAD software is a fully TMS320 DSP Algorithm Standard (XDAIS) compatible, reentrant code. The VAD interface complies with the TMS320 DSP Algorithm Standard and can be used in multitasking environments.

The TMS320 DSP Algorithm Standard (XDAIS) provides the user with object interface simulating object-oriented principles and asserts a set of programming rules intended to facilitate integration of objects into a framework.

The following documents provide further information regarding the TMS320 DSP Algorithm Standard (XDAIS):

- *Using the TMS320 DSP Algorithm Standard in a Static DSP System* (SPRA577)
- *TMS320 DSP Algorithm Standard Rules and Guidelines* (SPRU352)
- *TMS320 DSP Algorithm Standard API Reference* (SPRU360)
- *Technical Overview of eXpressDSP-Compliant Algorithms for DSP Software Producers* (SPRA579)
- *The TMS320 DSP Algorithm Standard* (SPRA581)
- *Achieving Zero Overhead with the TMS320 DSP Algorithm Standard IALG Interface* (SPRA716)

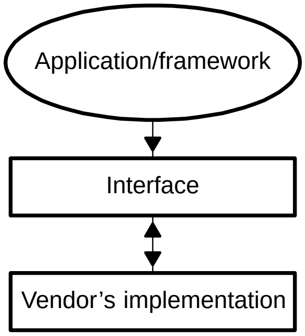
1.2 XDAIS Basics

This section instructs the user on how to develop applications/frameworks using the algorithms developed by vendors. It explains how to call modules through a fully eXpress DSP-compliant interface.

Figure 1-1 illustrates the three main layers required in an XDAIS system:

- Application/Framework layer
- Interface layer
- Vendor implementation. Refer to appendix A for a detailed illustration of the interface layer.

Figure 1-1. XDAIS System Layers



1.2.1 Application/Framework

Users should develop an application in accordance with their own design specifications. However, instance creation, deletion and memory management requires using a framework. It is recommended that the customer use the XDAIS framework provided by SPIRIT Corp. in ROM.

The framework in its most basic form is defined as a combination of a memory management service, input/output device drivers, and a scheduler. For a framework to support/handle XDAIS algorithms, it must provide the framework functions that XDAIS algorithm interfaces expect to be present. XDAIS framework functions, also known as the ALG Interface, are prefixed with “ALG_”. Below is a list of framework functions that are required:

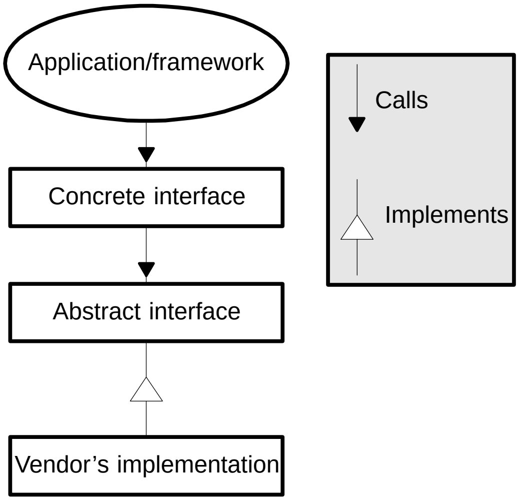
- `ALG_create` - for memory allocation/algorithm instance creation
- `ALG_delete` - for memory de-allocation/algorithm instance deletion
- `ALG_activate` - for algorithm instance activation

- `ALG_deactivate` - for algorithm instance de-activation
- `ALG_init` - for algorithm instance initialization
- `ALG_exit` - for algorithm instance exit operations
- `ALG_control` - for algorithm instance control operations

1.2.2 Interface

Figure 1-2 is a block diagram of the different XDAIS layers and how they interact with each other.

Figure 1-2. XDAIS Layers Interaction Diagram



1.2.2.1 Concrete Interface

A concrete interface is an interface between the algorithm module and the application/framework. This interface provides a generic (non-vendor specific) interface to the application. For example, the framework can call the function `MODULE_apply()` instead of `MODULE_VENDOR_apply()`. The following files make up this interface:

- Header file `MODULE.h` - Contains any required definitions/global variables for the interface.
- Source File `MODULE.c` - Contains the source code for the interface functions.

1.2.2.2 Abstract Interface

This interface, also known as the IALG Interface, defines the algorithm implementation. This interface is defined by the algorithm vendor but must comply with the XDAIS rules and guidelines. The following files make up this interface:

- Header file `iMODULE.h` - Contains table of implemented functions, also known as the IALG function table, and definition of the parameter structures and module objects.
- Source File `iMODULE.c` - Contains the default parameter structure for the algorithm.

1.2.2.3 Vendor Implementation

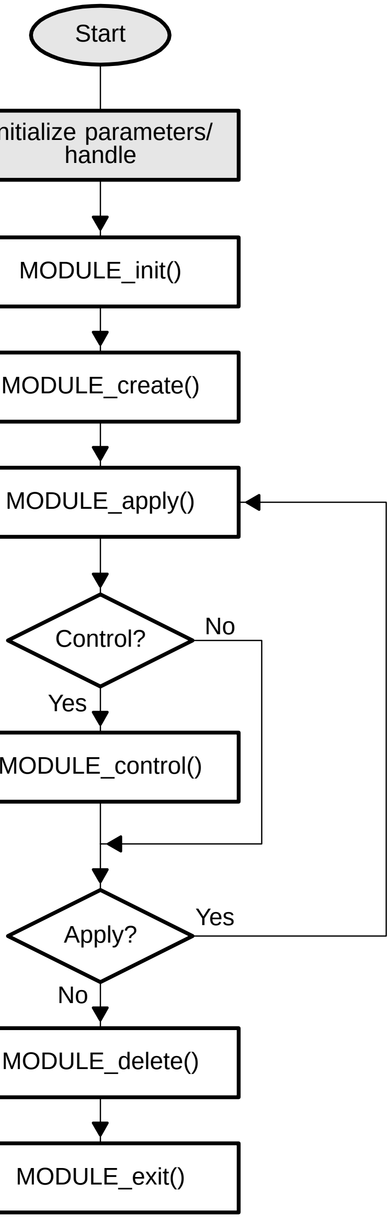
Vendor implementation refers to the set of functions implemented by the algorithm vendor to match the interface. These include the core processing functions required by the algorithm and some control-type functions required. A table is built with pointers to all of these functions, and this table is known as the function table. The function table allows the framework to invoke any of the algorithm functions through a single handle. The algorithm instance object definition is also done here. This instance object is a structure containing the function table (table of implemented functions) and pointers to instance buffers required by the algorithm.

1.2.3 Application Development

Figure 1-3 illustrates the steps used to develop an application. This flowchart illustrates the creation, use, and deletion of an algorithm. The handle to the instance object (and function table) is obtained through creation of an instance of the algorithm. It is a pointer to the instance object. Per XDAIS guidelines, software API allows direct access to the instance data buffers, but algorithms provided by SPIRIT prohibit access.

Detailed flow charts for each particular algorithm is provided by the vendor.

Figure 1-3. Module Instance Lifetime



The steps below describe the steps illustrated in Figure 1-3.

Step 1: Perform all non-XDAIS initializations and definitions. This may include creation of input and output data buffers by the framework, as well as device driver initialization.

Step 2: Define and initialize required parameters, status structures, and handle declarations.

Step 3: Invoke the `MODULE_init()` function to initialize the algorithm module. This function returns nothing. For most algorithms, this function does nothing.

Step 4: Invoke the `MODULE_create()` function, with the vendor's implementation ID for the algorithm, to create an instance of the algorithm. The `MODULE_create()` function returns a handle to the created instance. You may create as many instances as the framework can support.

Step 5: Invoke the `MODULE_apply()` function to process some data when the framework signals that processing is required. Using this function is not obligatory and vendor can supply the user with his own set of functions to obtain necessary processing.

Step 6: If required, the `MODULE_control()` function may be invoked to read or modify the algorithm status information. This function also is optional. Vendor can provide other methods for status reporting and control.

Step 7: When all processing is done, the `MODULE_delete()` function is invoked to delete the instance from the framework. All instance memory is freed up for the framework here.

Step 8: Invoke the `MODULE_exit()` function to remove the module from the framework. For most algorithms, this function does nothing.

The integration flow of specific algorithms can be quite different from the sequence described above due to several reasons:

- Specific algorithms can work with data frames of various lengths and formats. Applications can require more robust and effective methods for error handling and reporting.

- Instead of using `the MODULE_apply()` function, SPIRIT Corp. algorithms use extended interface for data processing, thereby encapsulating data buffering within XDAIS object. This provides the user with a more reliable method of data exchange.

1.3 Related Products

VAD objects can be efficiently used in conjunction with following products of Spirit Corp.:

- Automatic Gain Control
- Comfort Noise Generator
- Vocoders (G.721.3, G.729, G.726, G.711, proprietary low bit rate)

It is highly recommended to use this object in conjunction with SPIRIT Corp.'s Comfort Noise Generator (CNG), to regenerate noise with the same spectral shape, and Automatic Gain Control (AGC), to avoid gain adaptation during silence periods.

Since data format is very common, it can be easily interfaced with other third-party products.

Chapter 2

Voice Activity Detector (VAD) Integration

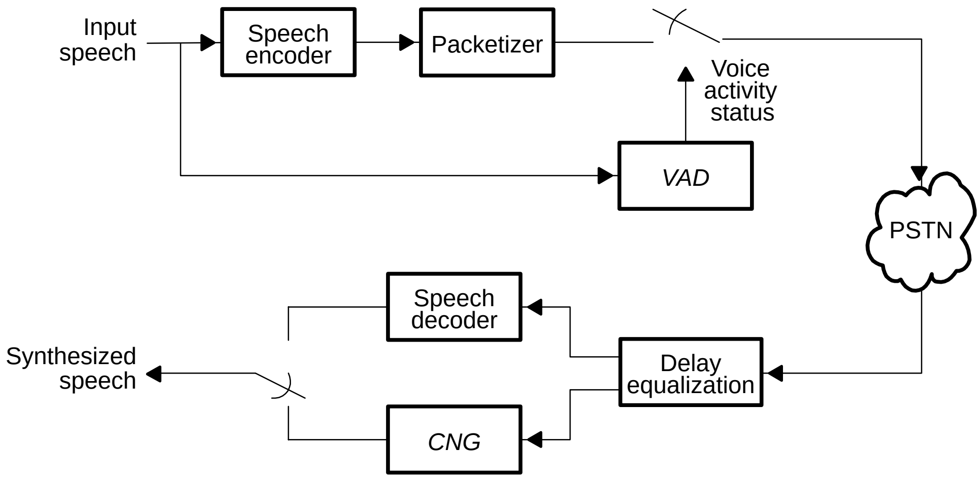
This chapter provides descriptions, diagrams, and example explaining the integration of the Voice Activity Detector (VAD) with frameworks.

Topic **Page**

2.1 Overview

Figure 2-1 illustrates a typical VAD integration diagram.

Figure 2-1. Packet Voice System with Voice Activity Detection (VAD) and Comfort Noise Generation (CNG)

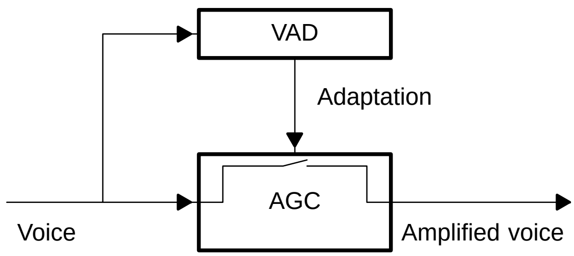


In brief, voice activity detection is performed over frames of input speech (typically per 30 msec). The goal of VAD is to determine all packets, which, if suppressed, could be reconstructed accurately by the comfort noise generator at the far end.

Note: Sampling Rate of Speech

It is assumed that the sampling rate of the speech is 8 kHz.

Figure 2-2. Using VAD to Control AGC Adaptation



VAD, with other parameters, can be used to control AGC adaptation. Disable the VAD on noise (silence) frames and enable it on voice (speech) frames.

The voice activity detector uses the following parameters in its decision process:

- Peak energy
- Minimum energy
- Prediction gain
- Average normalized squared pitch correlation. The maximum pitch correlation is averaged over a number of frames.
- Spectral non-stationarity

As a result, the detector indicates whether or not active speech is presentat that time.

2.2 Integration Flow

In order to integrate the VAD detector into a framework the user must follow these steps (Figure 2-3):

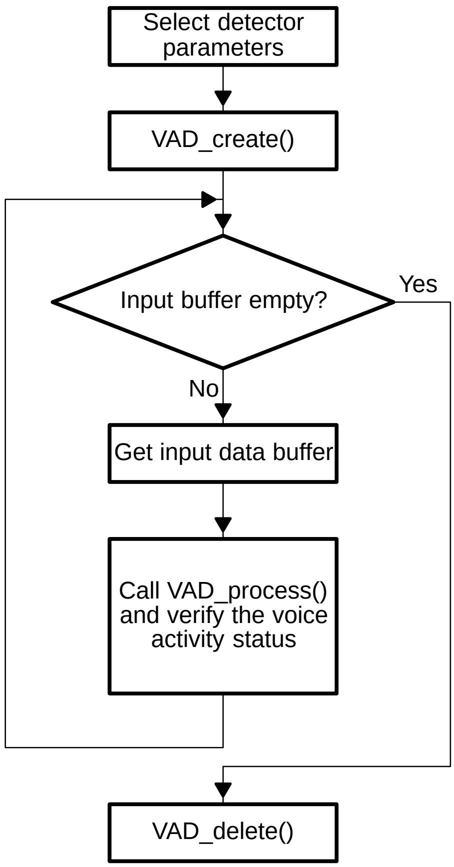
Step 1: Create the `VAD_Params` structure and initialize it with required values.

Step 2: Call `VAD_create()` to create the instance of a detector. There are no restrictions on the maximum number of detector instances created.

Step 3: Pass a stream with input samples (8 kHz, 16 bits) to `VAD_detect()` routine.

Step 4: Delete detector by using `VAD_delete()`.

Figure 2-3. Typical VAD Integration Flow



2.3 Example of a Call Sequence

The example below demonstrates a typical call sequence for the VAD detector. Full sample code is placed in file `Src\FlexExamples\Standalone XDAS\VAD\main.c`.

```
#include <stdio.h>
#include ”vad.h”
#include ”ivad.h”
#include ”vad_spcorp.h”
/* file I/O functions */
int fread16(const XDAS_Int16 *pData, size_t size, FILE *pFile);
int fwrite16(const XDAS_Int16 *pData, size_t size, FILE *pFile);
void main()
{
  FILE *inFile, *outFile;
  XDAS_Int16 pFrame[INPUT_FRAME]; /* buffer for input samples */
  XDAS_Int16 i;
  VAD_Handle VADInst;
  IVAD_Result vad;
  IVAD_NoiseLPCParams LPparam;
  printf(”SPIRIT VAD v 2.0\nprocessing in progress...\n”);
  /* open input and out files */
  inFile  = fopen(”VAD_demo.pcm”, ”rb”);
  outFile = fopen(”VAD_demo_out.pcm”, ”wb”);
  if(!inFile || !outFile)
  {
    printf(”file open error!\n”);
    return;
  }
  /* copy WAV header */
  fread(pFrame, 1, 44, inFile);
  fwrite(pFrame, 1, 44, outFile);
  /* create VAD instance with default parameters */
  VADInst = VAD_create(&VAD_SPCORP_IVAD, NULL);
  /* circle VAD processing */
  while(fread16(pFrame, INPUT_FRAME, inFile) == INPUT_FRAME)
```

```
  {
    /* process recieved samples */
    vad = VAD_process(VADInst, pFrame, INPUT_FRAME);
    /* check result */
    if(vad == IVAD_NOISE)
    {
      /* estimate LPC coefficients for noise generator */
      LPparam = VAD_getNoiseLPC(VADInst);
      /* run noise generator */
      . . . . . . . . . . .
    }
    else
    {
      /* do something if it's a not noise segment */
      . . . . . . . . . . .
    }
    /* do something in any case */
    . . . . . . . . . . .
    /* save processed samples */
    fwrite16(pFrame, INPUT_FRAME, outFile);
  }/* while */
  /* Deleting VAD instance */
  VAD_delete(VADInst);
  printf("\n...Finish\n");
  /* closing all opening files */
  fclose(inFile);
  fclose(outFile);
}
```

Chapter 3

Voice Activity Detector (VAD) API Descriptions

This chapter provides the user with a clear understanding of Voice Activity Detector (VAD) algorithms and their implementation with the TMS320 DSP Algorithm Standard interface (XDAIS).

Topic **Page**

3.1 Standard Interface Structures

The section describes parameter structures for the Voice Activity Detector.

Table 3-1 lists the type and location of the Standard Interface structures.

Table 3-1. Standard Interface Structures Summary

Parameters	Located in Table...
Voice Activity States	Table 3-2
Instance Creation	Table 3-3
Flags description	Table 3-4
VAD Real-time Status	Table 3-5

3.1.1 Voice Activity Status

Enumerated constants `VAD_Result` provide aliases for voice activity states.

Table 3-2. Voice Activity States

Enum	Voice Activity
`IVAD_NOISE`	Noise frame
`IVAD_NOISE_HANGOVER`	Marks a few number of first voice frames
`IVAD_VOICE`	Speech frame
`IVAD_VOICE_HANGOVER`	Arises after speech segment

3.1.2 Instance Creation Parameters

Description This structure defines the creation parameters for the algorithm. A default parameter structure is defined in "`iVAD.c`".

Structure Definition Use structure `IVAD_Params` to provide each instance with parameters.

Table 3-3. Interface Creation Parameters

`typedef struct IVAD_Params {`

Parameter Type	Parameter Name	Limits and Typical Value	Description
`XDAS_Int16`	`highAmp`	0...32767	Average speech amplitude; signal with higher amplitude will be recognized as speech.
`XDAS_Int16`	`lowAmp`	0...32767	Minimum noise amplitude; signal with lower amplitude will be recognized as noise.
`XDAS_Int16`	`quality`	-32768...32767	Used for VAD energetic thresholds adjustment. The lower this number is, the higher is probability that noise will be recognized as speech, and vice versa. In other words, these parameters allow to tune VAD for different speech-to-noise ratios or for different applications. If, for example, VAD is used together with vocoder, this parameter should be low (around –19000); in order to make sure that even noise-like speech segments will be detected as speech. On the other hand, if VAD is used to control AGC, this parameter can be chosen higher (above -10000), in order to make sure that only those segments that really represent speech are detected.
`XDAS_Int16`	`spFlat`	0...32767	Sets degree of closeness of background noise to broadband white noise; higher value corresponds to non-broadband noise.
`XDAS_Int16`	`updateCoef`	-32768...32767	Sets the degree of noise (stationary power spectrum envelope and of energy); decreasing the filed value corresponds to more stationary noise.
`XDAS_Int16`	`updateRate`	0...32767	Minimum energy update rate; it is used for checking noise adaptation correctness.
`XDAS_Int16`	`speechSmooth`	0...32767	Number of frames after speech frames will be marked as IVAD_VOICE_HANGOVER even if they contains noise (see).

Table 3-3. Interface Creation Parameters (Continued)

Parameter Type	Parameter Name	Limits and Typical Value	Description
`XDAS_Int16`	`noiseSmooth`	0...32767	Count of voice frames in a row before VAD result will be switched to IVAD_VOICE; before this VAD result will be IVAD_NOISE_HANGOVER (see Figure 3-1).
`XDAS_Int16`	`initFrames`	0...6	Power of two of frames, which must be used for noise estimation and adaptation; increasing of this value corresponds to decreasing adaptation speed.
`XDAS_Int16`	`flags`	0x0000...0xfff0	Sets modes of VAD by appropriate flags; the user must write “1” in the appropriate bit for enabling the needed mode; the numbering performed from low bit to high bit modes are listed in the table below.
`XDAS_Int16`	`cngOrder`	0...10	Number of LPC (Linear Prediction Coefficients); sets accuracy of spectrum envelope to be reproduced by CNG.
`XDAS_Int16`	`cngEnergyCoef`	0...32767	Eenergy smoothing for CNG.

```
} IVAD_Params
```

Figure 3-1. VAD Result Smoothing

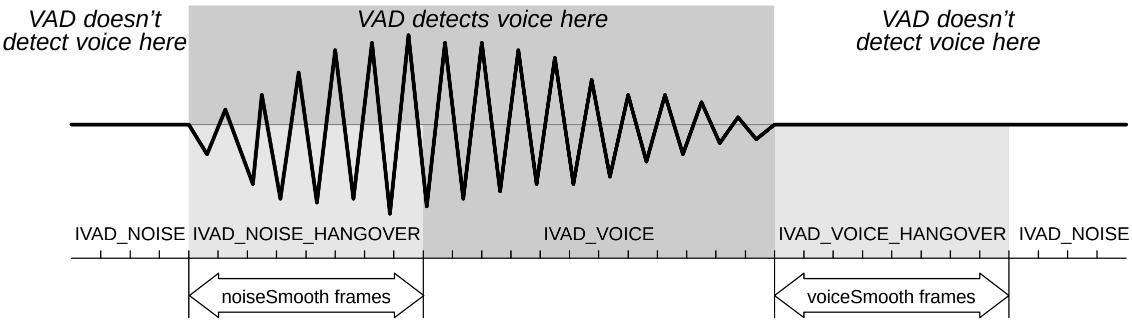


Table 3-4. Flags Description

Bit Number	Description
15	Enables/disables mode of continuous adaptation to noise level; it could be used in case of sudden noise changing; this flag is used only in case of enabled adaptation (bit 4 equal to “1”, see below).
14	Usage of *highAmp* field in decision making process
13	Usage of l*owAmp* field in decision making process
12	Usage of spectrum envelope stationarity in noise adaptation process
11	Usage of energy stationarity in noise adaptation process
10	Usage of closeness of background noise to broadband white noise in noise adaptation process
9	Usage of closeness of background noise to broadband white noise in decision making process
8-7	Usage of spectrum envelope stationarity in decision making process
6-5	Usage of energy stationarity in decision making process
4	Enables/disables adaptation mode to noise level;
3-0	Unused

Type `IVAD_Params` is defined in “`iVAD.h`”.

3.1.3 Status Structure

Structure Definition

Table 3-5. VAD Real-Time Status Parameters

Description Not used.

```
typedef struct IVAD_Status {
```

Status Type	Status Name	Description

```
} IVAD_Status;
```

Type `IVAD_Status` defined in “`iVAD.h`”.

3.2 Standard Interface Functions

Table 3-6 summarizes the standard Interface functions of the Voice Activity Detector API. They are required when using VAD.

`VAD_apply()` and VAD_`control()` are optional, but neither are supported by Spirit Corp.

Table 3-6. Echo Canceller Standard Interface Functions

Functions	Description	See Page...
`VAD_init`	Algorithm initialization	3-7
`VAD_exit`	Algorithm deletion	3-7
`VAD_create`	Instance creation	3-8
`VAD_delete`	Instance deletion	3-8

3.2.1 Algorithm Initialization

VAD_init *Calls the framework initialization function to initialize an algorithm*

Description This function calls the framework initialization function, `ALG_init()`, to initialize the algorithm. For VAD detector, this function does nothing. It can be skipped and removed from the target code according to *Achieving Zero Overhead With the TMS320 DSP Algorithm Standard IALG Interface* (SPRA716).

Function Prototype `void VAD_init()`

Arguments none

Return Value none

3.2.2 Algorithm Deletion

VAD_exit *Calls the framework exit function to remove an algorithm*

Description This function calls the framework exit function, `ALG_exit()`, to remove the algorithm. For VAD detector, this function does nothing. It can be skipped and removed from the target code according to *Achieving Zero Overhead With the TMS320 DSP Algorithm Standard IALG Interface* (SPRA716).

Function Prototype `void VAD_exit()`

Arguments none

Return Value none

3.2.3 Instance Creation

VAD_create *Function called in order to creat a new VAD detector object*

Description

In order to create a new VAD detector object, `VAD_create` function should be called. This function calls the framework create function, `ALG_create()`, to create the instance object and perform memory allocation tasks. Global structure `VAD_SPCORP_IVAD` contains `VAD` virtual table supplied by SPIRIT Corp.

Function Prototype

```
VAD_Handle VAD_create
      (const IVAD_Fxns *fxns,
       const VAD_Params *prms);
```

Arguments

`IVAD_Fxns *`	Pointer to vendor's functions (Implementation ID). Use reference to `VAD_SPCORP_IVAD` virtual table supplied by SPIRIT Corp.
`VAD_Params *`	Pointer to Parameter Structure. Use `NULL` pointer to load default parameters.

Return Value

`VAD_Handle` Defined in file "`VAD.h`". This is a pointer to the created instance.

3.2.4 Instance Deletion

VAD_delete *Calls the framework delete function to delete an instance object*

Description

This function calls the framework delete function, `ALG_delete()`, to delete the instance object and perform memory de-allocation tasks.

Function Prototype

```
void VAD_delete (VAD_Handle handle)
```

Arguments

`VAD_Handle` Instance's handle obtained from `VAD_create()`

Return Value

none

3.3 Vendor-Specific Interface

This section describes the vendor-specific functions in the SPIRIT's algorithm implementation and interface (extended IALG methods). Table 3-7 summarizes SPIRIT's API functions of VAD detector.

The whole interface is placed in header files `iVAD.h`, `VAD.h`, `VAD_spcorp.h`.

Table 3-7. Detector-Specific Interface Functions

Functions	Description	See Page...
`VAD_reInit`	Sets VAD parameters	3-9
`VAD_process`	Returns valid status of voice activity	3-10
`VAD_getNoiseLPC`	Returns structure with LPC coefficients for CNG	3-10

3.3.1 Re-initialization

VAD_reInit *Returns valid call progress tones or special notification messages*

Description Returns valid call progress tones or special notification message.

Function Prototype

```
IVAD_Symbol VAD_reInit
      (VAD_Handle handle,
       IVAD_Params * params)
```

Arguments

`handle` Pointer to VAD instance

`*params` Pointer to parameter structure

Return Value none

Restrictions none

3.3.2 Voice Dctivity Detection

VAD_process *Returns the valid voice activity status*

Description Returns valid voice activity status.

Function Prototype

```
XDAS_Result VAD_process
      (VAD_Handle handle,
       XDAS_Int16 *pBuf,
       XDAS_Int16 bufSize)
```

Arguments

`handle`	Pointer to VAD instance
`*pBuf`	Array of input samples at sample rate 8 kHz
`bufSize`	Number of samples to be processed

Return Value Voice activity status according to Table 3-2.

Restrictions Maximum length of input and output buffer is 65535.

3.3.3 Get LPC for CNG

VAD_getNoiseLPC *Performs an estimation of LPC coefficients for the noise generator*

Description Performs an estimation of LPC coefficients for the noise generator.

Function Prototype

```
IVAD_NoiseLPCParams VAD_getNoiseLPC
      (VAD_Handle handle)
```

Arguments

`handle`	Pointer to VAD instance

Used Definition

```
typedef struct IVAD_NoiseLPCParams {
```

Status Type	Status Name	Description
XDAS_Int16	pLPC[IVAD_LPCORDER]	buffer for LPC coefficients
XDAS_Int16	order	number of LPC coefficients
XDAS_Int16	resAmp	residual amplitude

```
} IVAD_NoiseLPCParams;
```

Return Value Actual detector status (see Table 3-5).

Restrictions none

Appendix A

Test Environment

Note: Test Environment Location

This chapter describes test environment for the VAD object.

For TMS320C54CST device, test environment for standalone VAD object is located in the Software Development Kit (SDK) in `Src\FlexExamples\StandaloneXDAS\VAD`.

Topic **Page**

A.1 Description of Directory Tree

The SDK package includes the test project “test.pjt” and corresponding reference test vectors. The user is free to modify this code as needed, without submissions to SPIRIT Corp.

Table A-1. Test Files for VAD

File	Description
main.c	Test file
FileC5x.c	File input/output functions
..\ROM\CSTRom.s54	ROM entry address
Test.cmd	Linker command file
Vectors\output.pcm	Reference output test vectors

A.1.1 Test Vectors Format

All test vectors are raw PCM files with following parameters:

- Bits per sample - 16, Mono
- Word format - Intel PCM (LSB goes first)
- Encoding - Uniform

A.1.2 Test Project

To build and run a project, the following steps must be performed:

Step 1: Open the project: `Project\Open`

Step 2: Build all necessary files: `Project\Rebuild All`

Step 3: Initialize the DSP: `Debug\Reset CPU`

Step 4: Load the output-file: `File\Load program`

Step 5: Run the executable: `Debug\Run`

Once the program finishes testing, the file *Output.pcm* will be written in the current directory. Compare this file with the reference vector contained in the directory *Vectors*.

Note: Test Duration

Since the standard file I/O for EVM is very slow, testing may take several minutes. Test duration does not indicate the real algorithm's throughput.

Index